Métodos de Ordenação

Profª. Barbara Quintela
Prof. Jose J. Camata
Prof. Marcelo Caniato

barbara@ice.ufjf.br
camata@ice.ufjf.br
marcelo.caniato@ice.ufjf.br

Conceitos Básicos

- **Ordenar**: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.

Conceitos Básicos

- Classificação dos métodos de ordenação:
  - **Interna**: arquivo a ser ordenado cabe todo na memória principal.
  - **Externa**: arquivo a ser ordenado não cabe na memória principal.
- Diferenças entre os métodos:
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.
  - Em um método de ordenação externa, os registros são acessados sequencialmente ou em grandes blocos.
- A maioria dos métodos de ordenação é baseada em **comparações das chaves**.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo *n* o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações *C(n)* entre chaves.
  - Número de movimentações *M(n)* de itens do arquivo.

Ordenação Interna

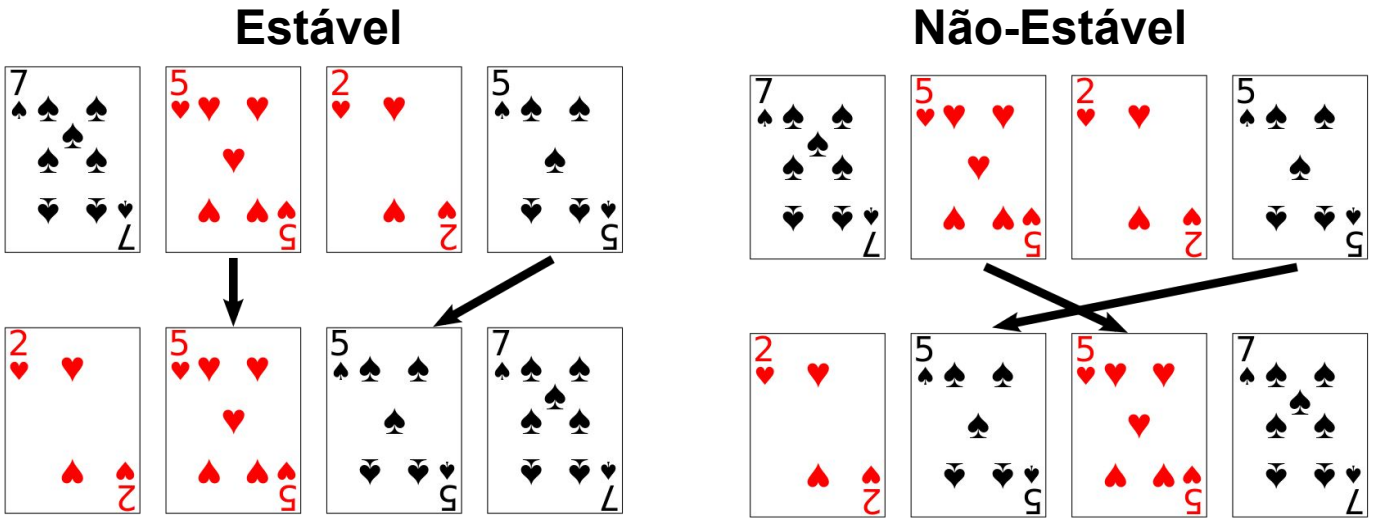
- O uso econômico da memória disponível é um requisito primordial na ordenação interna.
- Métodos de ordenação i*n situ* são os preferidos.
- Métodos que fazem cópias dos itens a serem ordenados possuem menor importância.

Ordenação Interna

- Classificação dos métodos de ordenação interna:
  - **Métodos simples:**
    - Adequados para pequenos arquivos.
    - Requerem $O(n^2)$ comparações.
    - Produzem programas pequenos.
  - **Métodos eficientes:**
    - Adequados para arquivos maiores.
    - Requerem $O(n \log n)$ comparações.
    - Usam menos comparações.
    - As comparações são mais complexas nos detalhes.
    - Métodos simples são mais eficientes para pequenos arquivos.

Ordenação Estável *vs* Não Estável

- Um algoritmo de **ordenação** diz-se **estável** se preserva a ordem de registros de chaves iguais.
- Útil apenas quando há dados associados às chaves de ordenação.

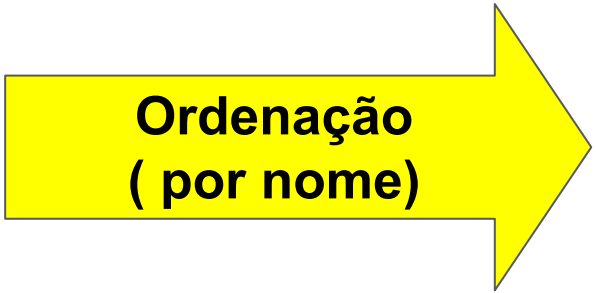


Ordenação Estável - Exemplo

Considere o seguinte conjunto de dados:

- Nomes dos alunos e suas respectivas turmas.

(Davi, A)
(Aline, B)
(Jose, A)
(Eduardo, B)
(Carolina, A)

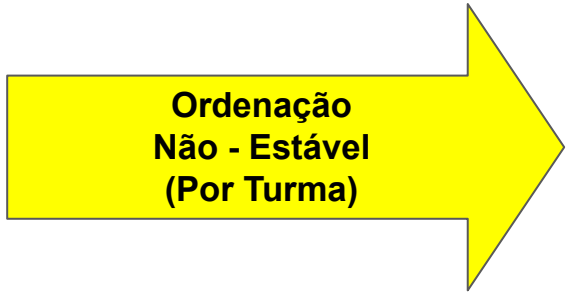


(Aline, B)
(Carolina, A)
(Davi, A)
(Eduardo, B)
(Jose, A)

Ordenação Estável - Exemplo

Agora vamos considerar reordenar a lista resultante em relação às turmas usando **algoritmo não estável.**

(Aline, B)
(Carolina, A)
(Davi, A)
(Eduardo, B)
(Jose, A)



(Carolina, A)
(Jose, A)
(Davi, A)
(Eduardo, B)
(Aline, B)

Ordenação Estável - Exemplo

Agora vamos considerar reordenar a lista resultante em relação às turmas usando **algoritmo estável.**

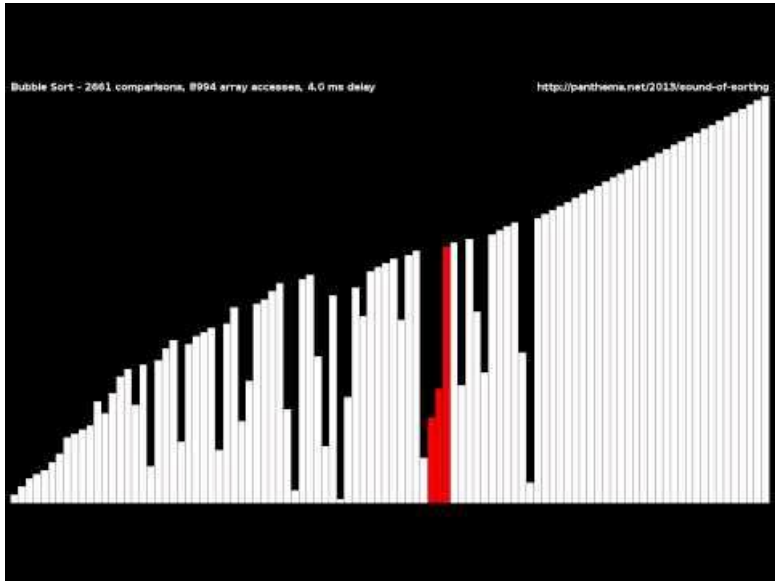
(Aline, B)
(Carolina, A)
(Davi, A)
(Eduardo, B)
(Jose, A)

(Carolina, A)
(Davi, A)
(Jose, A)
(Aline, B)
(Eduardo, B)

Método BubbleSort (Bolha)

Método BubbleSort

- Ideia básica:
  - Comparam-se dois elementos e trocam-se suas posições se o segundo elemento é menor do que o primeiro
  - São feitas várias passagens pelos registros
  - Em cada passagem, comparam-se dois elementos adjacentes
  - Se estes elementos estiverem fora de ordem, eles são trocados



O Método BubbleSort Ilustrado

L[0] > L[1] (F) → Não Troca

L[1] > L[2] (V) → Troca

L[2] > L[3] (V) → Troca

L[3] > L[4] (V) → Troca

L[4] > L[5] (V) → Troca

O	R	D	E	N	A
O	R	D	E	N	A
O	**R**	D	E	N	A
O	D	**R**	E	N	A
O	D	E	**R**	N	A
O	D	E	N	**R**	A
O	D	E	N	A	R

O Método BubbleSort Ilustrado

L[0] > L[1] (V) → Troca

L[1] > L[2] (V) → Troca

L[2] > L[3] (V) → Troca

L[3] > L[4] (V) → Troca

O	D	E	N	A	R
O	D	E	N	A	R
D	O	E	N	A	R
D	E	O	N	A	R
D	E	N	O	A	R
D	E	N	A	O	R

O Método BubbleSort Ilustrado

L[0] > L[1] (V) → Não Troca

L[1] > L[2] (V) → Não Troca

L[2] > L[3] (V) → Troca

D	E	N	A	O	R
D	E	N	A	O	R
D	E	N	A	O	R
D	E	N	A	O	R
D	E	A	N	O	R

O Método BubbleSort Ilustrado

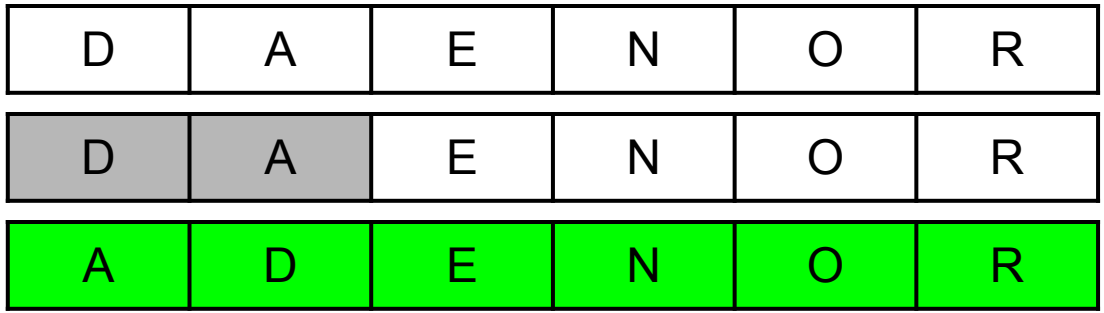
L[0] > L[1] (V) → Não Troca

L[1] > L[2] (V) → Troca

D	E	A	N	O	R
D	E	A	N	O	R
D	E	A	N	O	R
D	A	E	N	O	R

O Método BubbleSort Ilustrado

L[0] > L[1] (V) → Troca



Método BubbleSort

- Vantagens
  - Simplicidade do algoritmo
  - Estável
- Indicações
  - Tabelas muito pequenas
  - Quando se sabe que a tabela está quase ordenada
  - Demonstrações didáticas
- Desvantagens
  - Lentidão
- Origem da denominação
  - Os elementos menores (mais “leves”) vão aos poucos “subindo” para o início da tabela, como se fossem bolhas

Algoritmo BubbleSort

```
(1)  BubbleSort(L: vetor, n: inteiro)
(2)  Para i ← n-1 até 1 faça
(3)     Para j ← 1 até i faça
(4)         Se L[j] > L[j+1] Então
(5)             troca(L[j],L[j+1])
(6)          Fim-se
(7)     Fim-Para
(8)  Fim-Para
```

Observações:

- Número de comparações entre chaves de registros, pior caso: $C(n) = (n^2 - n)/2$.
- Custo: $O(n^2)$
- Algoritmo estável

A seguir:

Seleção, Inserção e MergeSort

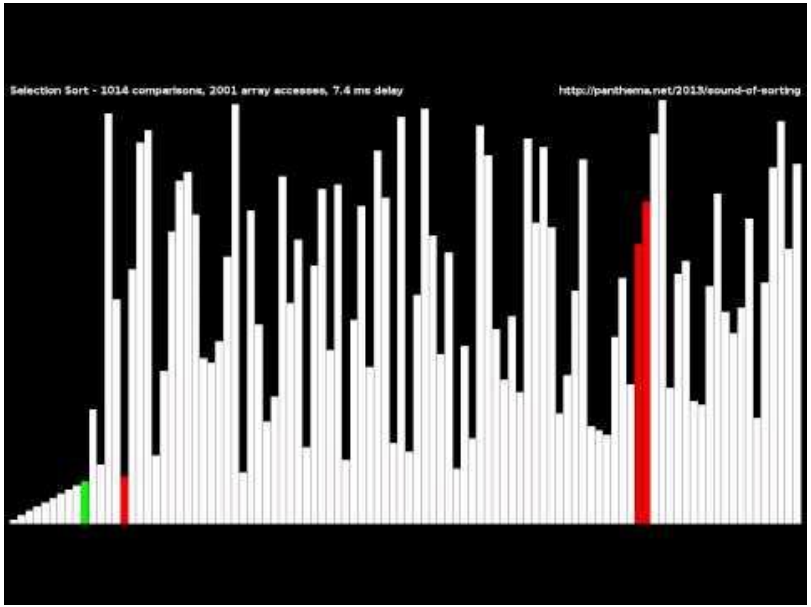
Métodos de Ordenação:

Seleção, Inserção e MergeSort

Ordenação por Seleção

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação.
- Ideia básica:
  - Selecione o menor item do vetor.
  - Troque-o com o item da primeira posição do vetor.
  - Repita essas duas operações com os n − 1 itens restantes, depois com os n − 2 itens, até que reste apenas um elemento



O método de Seleção Ilustrado...

Menor valor na posição 5:
Troca L[0] com L[5]

Menor valor na posição 2
Troca L[1] com L[2]

Menor valor na posição 3
Troca L[2] com L[3]

Menor valor na posição 4
Troca L[3] com L[4]

Menor valor na posição 5
Troca L[4] com L[5]

0	1	2	3	4	5
O	R	D	E	N	A
O	R	D	E	N	**A**
A	R	**D**	E	N	O
A	D	R	**E**	N	O
A	D	E	R	**N**	O
A	D	E	N	R	O
A	D	E	N	O	R

Ordenação por Seleção

```
(1)  SelectSort(L:vetor, n: inteiro)
(2)  Para i ← 1 até n-1 faça
(3)      min = i
(4)      Para j ← i+1 até n faça
(5)          Se L[j] < L[min] min = j
(6)      Fim-Para
(7)      troca(L[i],L[min]);
(8)  Fim-Para
```

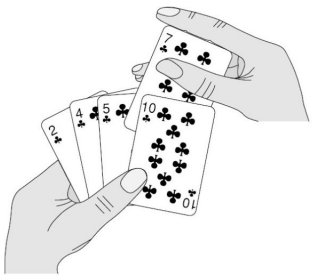
Ordenação por Seleção

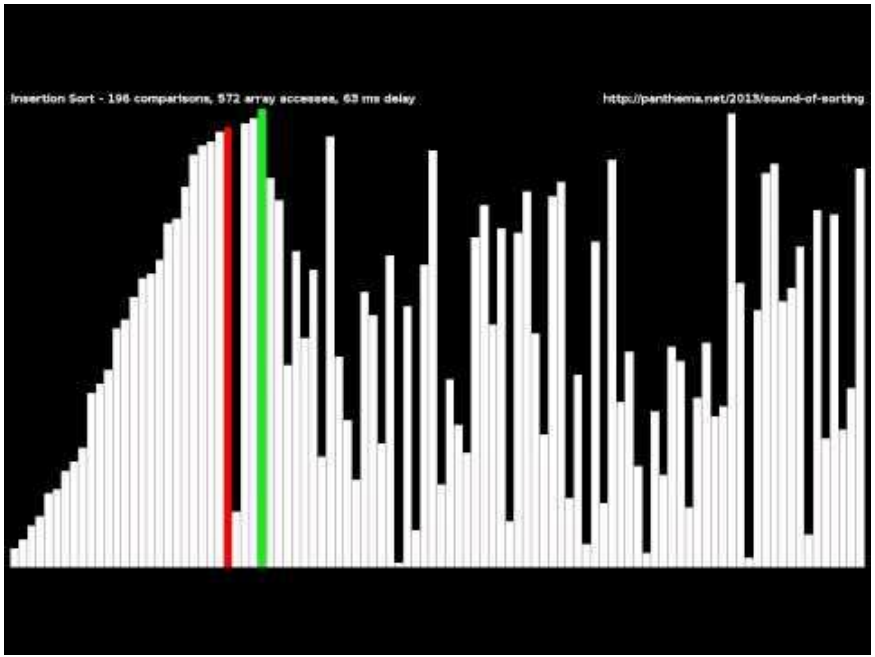
- Vantagens:
  - **Custo linear** para o número de **movimentos de registros.**
  - É o algoritmo a ser utilizado para arquivos com registros muito grandes.
  - É muito interessante para arquivos pequenos
- Desvantagens:
  - Custo Assintótico: $O(n^2)$
  - O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.

Ordenação por Inserção

Ordenação por Inserção

- Método preferido dos jogadores de cartas.
- Ideia Básica:
  - Em cada passo a partir de i=2 faça:
    - Selecione o i-ésimo item da sequência fonte.
    - Coloque-o no lugar apropriado na seqüência destino de acordo com o critério de ordenação.





O método de Inserção Ilustrado..

O é primeiro elemento na lista ordenada.

Escolhe L[2] (R) como pivô e insere na lista após o O

Escolhe L[3] (D) como pivô e insere na lista antes do O

Escolhe L[4] (E) como pivô e insere na lista antes do D

Escolhe L[5] (N) como pivô e insere na lista antes do O

Escolhe L[6] (N) como pivô e insere na lista antes do D

1	2	3	4	5	6
O	R	D	E	N	A
O	**R**	D	E	N	A
O	R	**D**	E	N	A
D	O	R	**E**	N	A
D	E	O	R	**N**	A
D	E	N	O	R	**A**
A	D	E	N	O	R

Ordenação por Inserção

```
(1)  InsertSort(L: vetor, n: inteiro)
(2)  Para j ← 2 até n faça
(3)      pivo ← L[j];
(4)      i = j - 1;
(5)      Enquanto (i > 0) E (L[i] > pivo)
(6)          L[i+1] = L[i];
(7)          i = i - 1;
(8)      Fim-Enquanto
(9)      L[i+1] = pivo
(10) Fim-Para
```

Ordenação por Inserção

- Vantagens
  - Simplicidade do algoritmo
  - Duas vezes mais rápido que da Bolha e normalmente mais rápido que a Seleção
  - Estável
- Desvantagens
  - Ainda é considerado um algoritmo lento
  - Custo assintótico pior caso: $O(n^2)$ (itens estão originalmente na ordem reversa)

Ordenação por Inserção

- Indicações
  - É o método a ser utilizado quando o arquivo está “quase” ordenado.
  - É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado, pois o custo é linear.

Estudo da Estabilidade

- O algoritmo é considerado estável, pois não há a possibilidade de elementos iguais mudar de posição no processo de ordenação

	4 6 7 2 9 8 4 1 3 4 0
Iteração 1	**4 6 | 7 2 9 8 4 1 3 4 0**
Iteração 5	**2 4 6 7 8 9 | 4 1 3 4 0**
Iteração 6	**2 4 4 6 7 8 9| 1 3 4 0**
Iteração 9	**1 2 3 4 4 4 6 7 8 9 | 0**

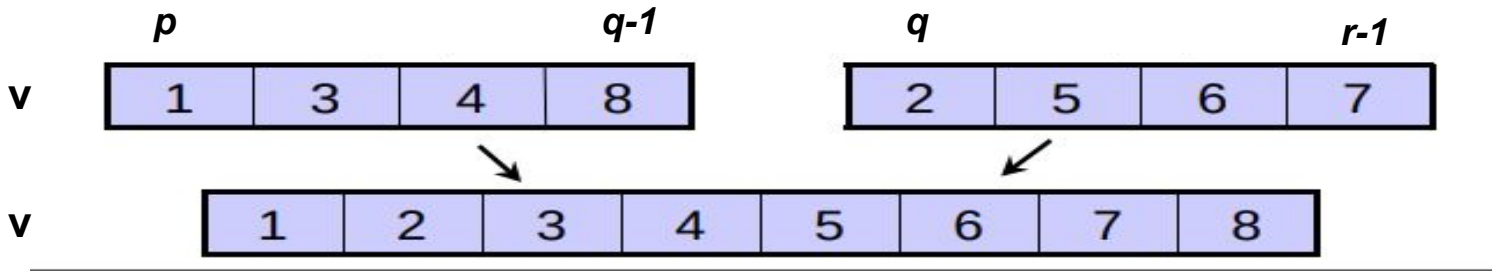
Ordenação por Intercalação (MergeSort)

Ordenação por Intercalação

- Ideia Básica
  - intercalar duas metades da lista desejada quando estas já se encontram ordenadas.
- Aplica um método “dividir para conquistar”
  - **divisão:** divide a lista em duas metades
  - **conquista**: ordena recursivamente cada sub-lista (dividindo novamente, quando possível) com a ordenação de intercalação.
  - **combina**: faz o intercalação das duas sub-listas ordenados para obter lista ordenada completamente.

Algoritmo de Intercalação: Merge

- **Intercalação:** Dados vetores crescentes v[*p .. q-1*] e v[*q .. r-1*], rearranjá-los um vetor v[*p .. r-1*] em ordem crescente.



Algoritmo Merge

```
(1)  merge(v:vetor, p: int, q: int, r: int)
(2)       i ← p
(3)       j ← q
(4)       k ← 0
(5)       Enquanto (i < q) E (j < r) Faça
(6)          Se v[i] < v[j] ) Então
(7)               aux[k] ← v[i]; i++;
(8)          Caso Contrário
(9)               aux[k] ←  v[j];j++;
(10)         Fim-Se
(11)         k++;
(12)      Fim-Enquanto
(13)      Enquanto( i < q) aux[k] = v[i]; i++; k++;
(14)      Enquanto (j < r) aux[k] = v[j]; j++; k++;
(15)      Para i ← p até r faça
(16)           v[i] = aux[i-p]
(17)      Fim-Para
(18)  Fim-procedimento
```

Observação:

(1) Utiliza um vetor temporário aux para manter o resultado da ordenação das duas sub-listas.

(2) Custo linear

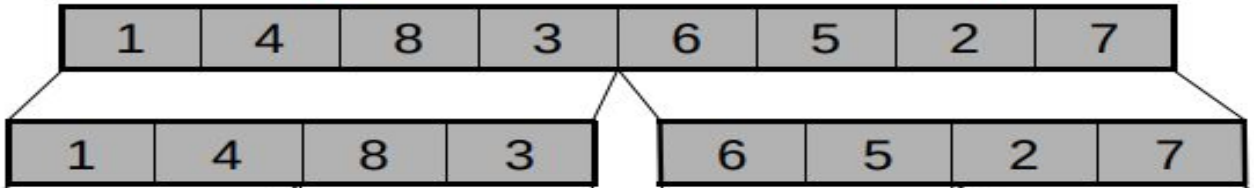
Algoritmo MergeSort

- Algoritmo é executado de forma recursiva
  - Passo 1: Divida o vetor ao meio
  - Passo 2: Ordenar as duas metade
  - Passo 3: intercalar as duas metades.

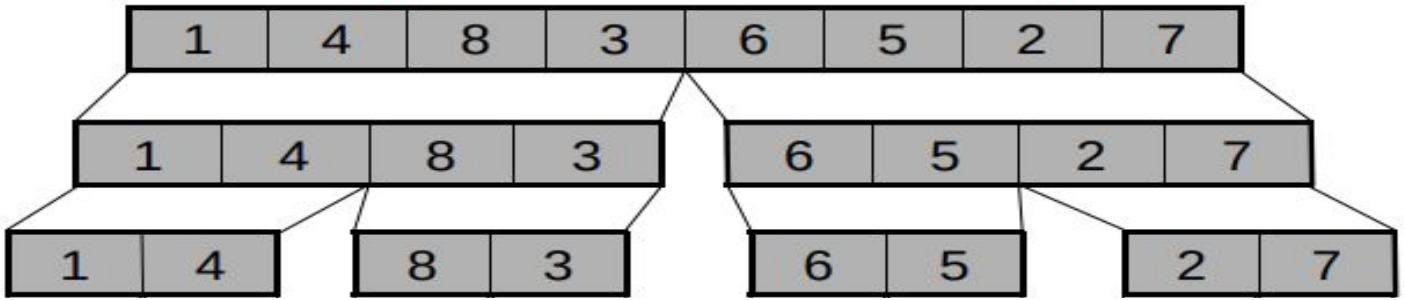
```
(1)  mergeSort(v:vetor, p:int, r: int)
(2)       Se p < (r-1) então
(3)            q = (p + r)/2;
(4)            mergeSort(v, p , q);
(5)            mergeSort(v, q , r);
(6)            merge(L,p,q,r);
(7)        Fim-Se
(8)  Fim-procedimento
```

MergeSort - Exemplo

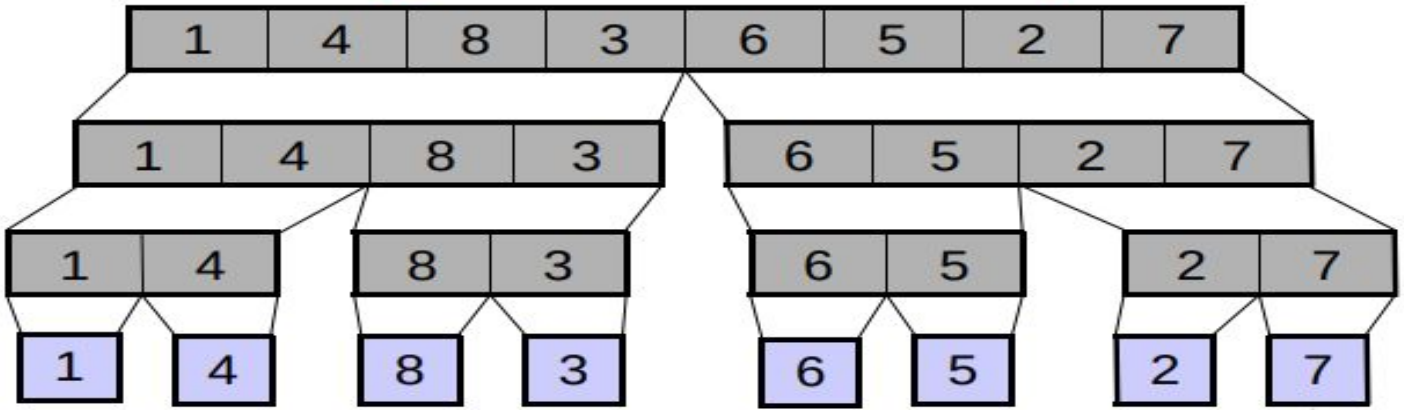
MergeSort - Exemplo



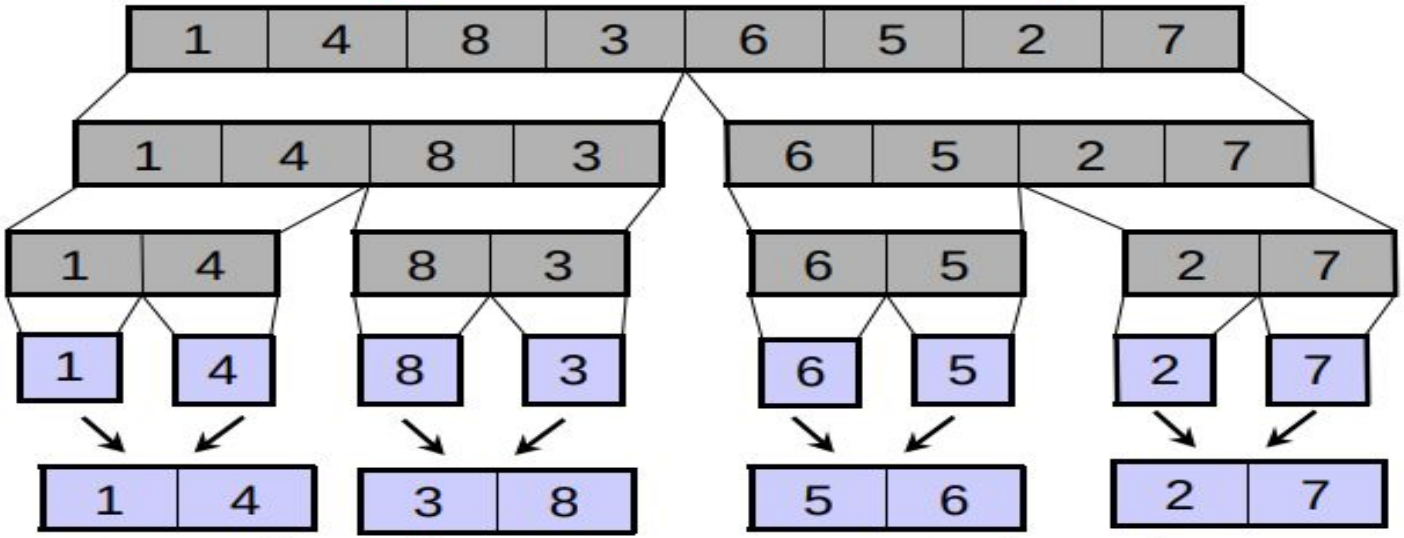
MergeSort - Exemplo



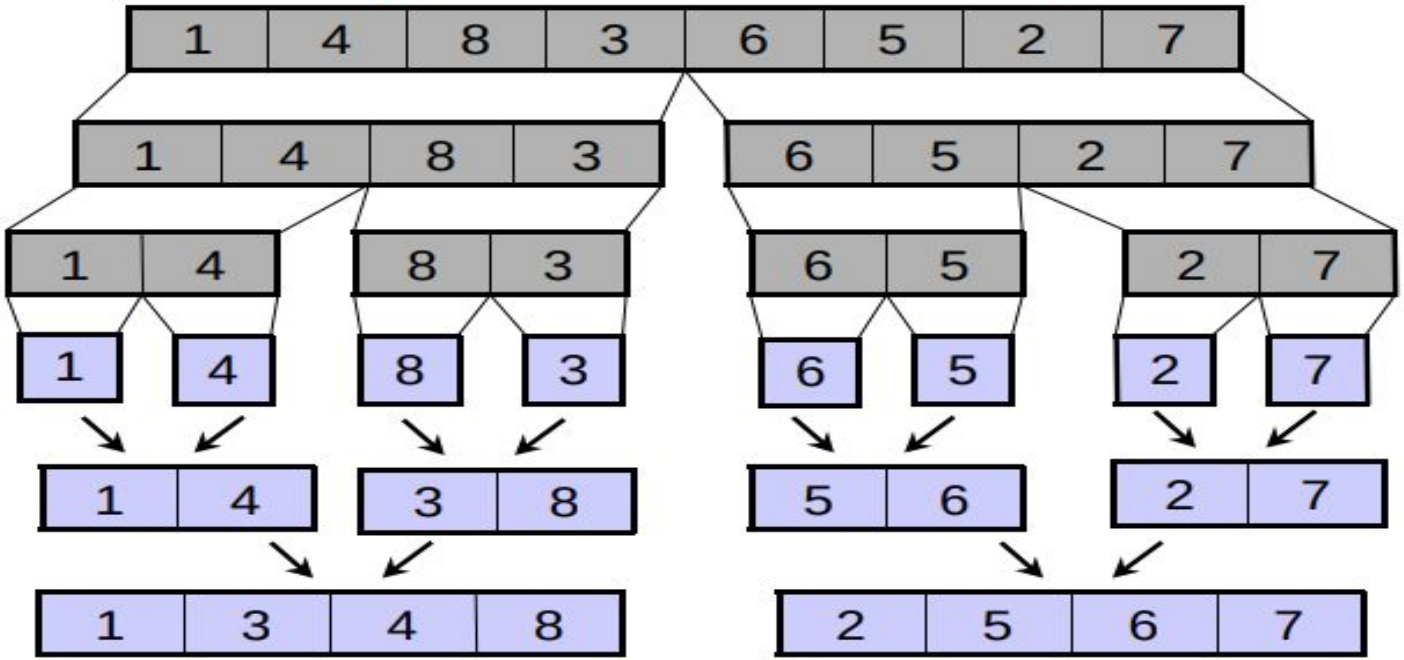
MergeSort - Exemplo



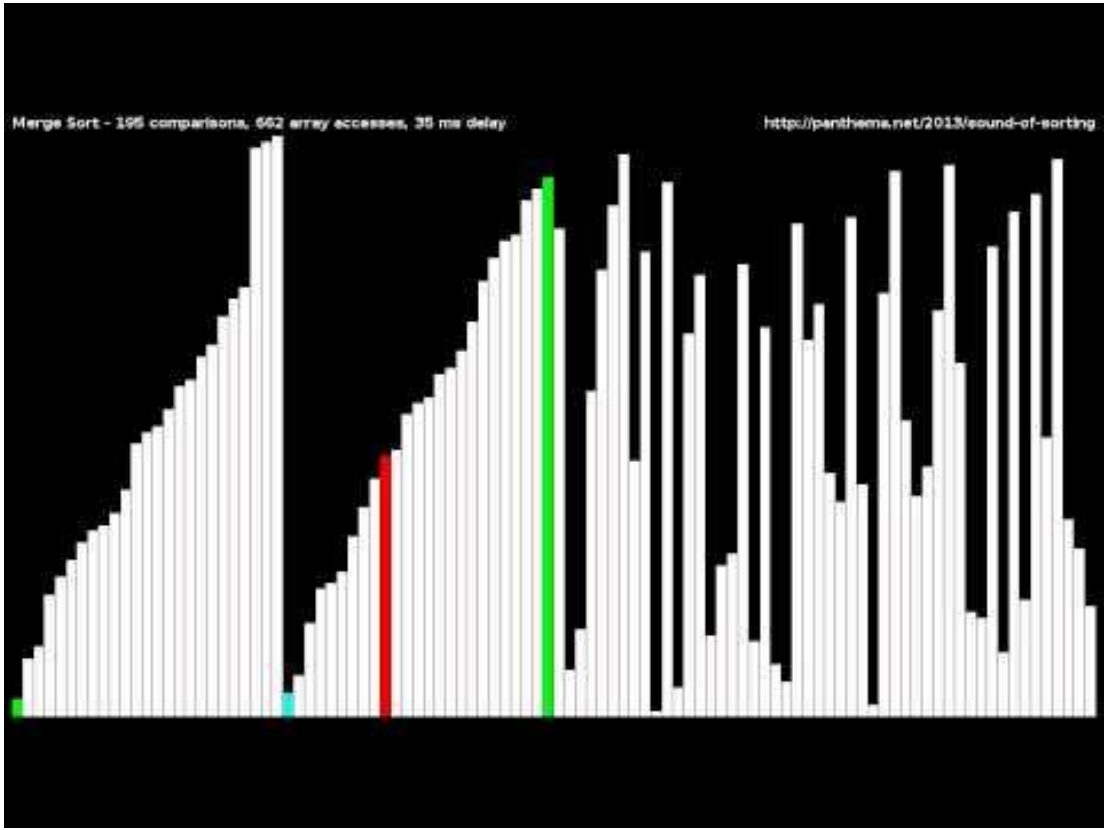
MergeSort - Exemplo



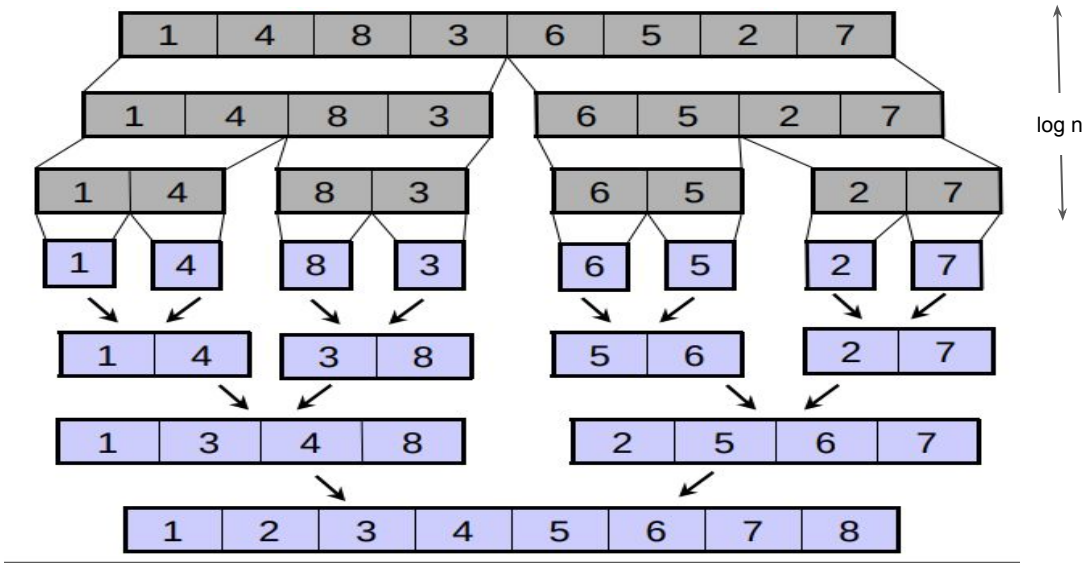
MergeSort - Exemplo



MergeSort Ilustrado...



Análise de desempenho



Observações:

- O número total de rodadas é aproximadamente log n
- Em cada rodada, a função intercala executa 2n movimentações de elementos
- Custo: O(nlogn)

Estudo de Estabilidade

- O **algoritmo** é considerado **estável**, pois não há a possibilidade de elementos iguais mudar de posição no processo de ordenação
- A fase de divisão do algoritmo não altera a posição de nenhuma chave
- **Ordenação é feita pelo algoritmo de intercalação**
- A intercalação é feita verificando, sequencialmente, os elementos de acordo com sua posição no vetor.
- Dessa forma, elementos com mesma chave não terão a sua posição relativa alterada

Próxima aula:

QuickSort e HeapSort

Exercício

1. Faça um teste de mesa com cada método de ordenação estudado até o momento, utilizando as seguintes sequências de dados de entrada:
   a. S1 = {2, 4, 6, 8, 10, 12 };
   b. S2 = {5, 7, 2, 8, 1, 6};

Em cada caso, mostre o número de comparações e trocas que realizam na ordenação de sequências.